Approximation Algorithms for Generalized Path Scheduling

Haozhou Pang
Department of Computer Science, University of Alberta, Edmonton, Canada
haozhou@ualberta.ca

Mohammad R. Salavatipour
Department of Computer Science, University of Alberta, Edmonton, Canada
mrs@ualberta.ca

Abstract

Scheduling problems where the machines can be represented as the edges of a network and each job needs to be processed by a sequence of machines that form a path in this network have been the subject of many research articles (e.g. flow shop is the special case where the network as well as the sequence of machines for each job is a simple path). In this paper we consider one such problem, called Generalized Path Scheduling (GPS) problem, which can be defined as follows. Given a set of non-preemptive jobs J and identical machines M ($|J| = n$ and $|M| = m$). The machines are ordered on a path. Each job $j = \{P_j = \{l_j, r_j\}, p_j\}$ is defined by its processing time p_j and a sub-path P_j from machine with index l_j to r_j ($l_j, r_j \in M$, and $l_j \leq r_j$) specifying the order of machines it must go through. We assume each machine has a queue of infinite size where jobs can sit in the queue to resolve conflicts. Two objective functions, makespan and total completion time, are considered. Machines can be identical or unrelated. In the latter case, this problem generalizes the classical Flow shop problem (in which all jobs have to go through all machines from 1 to m in that order).

Generalized Path Scheduling has been studied (e.g. see [9, 4]). In this paper, we present several improved approximation algorithms for both objectives. For the case of number of machines being sub-logarithmic in the number of jobs we present a PTAS for both makespan and total completion time. The PTAS holds even on unrelated machines setting and therefore, generalizes the result of Hall [7] for the classic problem of Flow shop. For the case of identical machines, we present an $O(\frac{\log m}{\log \log m})$-approximation algorithms for both objectives, which improve the previous best result of [4]. We also show that the GPS problem is NP-complete for both makespan and total completion time objectives.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Approximation Algorithms, Path Scheduling, Flow shop, Job Shop

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2020.10

Related Version Full version available at `https://webdocs.cs.ualberta.ca/~mreza/research/gen-flow.pdf`.

Funding *Haozhou Pang*: Supported by NSERC and Department of Computing Science at University of Alberta.
Mohammad R. Salavatipour: Supported by NSERC.

1 Introduction

Scheduling problems are well-studied over the last several decades due to their applications in various fields (from Operations Research, to Computer Science). One of the most classical scheduling problem is Job Shop: Given a set J of n jobs and a set M of m machines. Each job j consists of a sequence of λ_j operations $O_{1,j}, O_{2,j}, \ldots, O_{\lambda_j,j}$. The amount of time that job j takes to complete its operation $O_{i,j}$ on machine $M_i \in M$ is denoted as $p_{i,j}$. The goal of the problem is to find a feasible schedule that satisfies all constraints, while trying to optimize some objective function. In a feasible schedule, no machine can process more than one job at any time and each job can be run on at most one machine at any time. More constraints can be added depending on specific interests. For example, one can enforce that operations of a job need to be processed in a specific order (known as precedence constraints), so an operation cannot be processed until all its preceding operations are finished. The scheduling problems have drawn much attention because of their wide applications in many day-to-day situations. As an example, think of the given machines as routers and jobs as messages to be sent from one router to another through a specified path. A good scheduling algorithm can be applied here to send all messages efficiently.

In this paper we consider a variant of job-shop that we call Generalized Path Scheduling. This problem in the most general setting of unrelated machines generalizes Flow shop and it has been studied by [4, 9] among others.

▶ **Definition 1** (Generalized Path Scheduling). *Given a set of non-preemptive jobs J and identical machines M ($|J| = n$ and $|M| = m$) that form a path (edges representing the machines), each job j has a processing time p_j and a sub-path P_j of machines from machine l_j to r_j ($l_j \leq r_j$) specifying the order of machines it must go through. Each machine has a queue where jobs can wait in before being processed on that machine. We consdier minimizing makespan (largest completion time) and/or total completion time (also called min-sum).*

If we have unrelated machines but all the jobs have to go through all the machines in the same order then we have the classic Flow shop problem [17]. Authors of [4] introduced a more general setting in which the network of machines is a general graph (instead of a path) and each job has a path (sequence of machines) in this graph to go through. Several special cases of this problem have been studied before (see below). The two objective functions makespan and total completion time that we consider in this paper have been studied extensively in the literature. Let C_j be the completion time of job j (the time when j finishes its last operation) in a given schedule. The makespan of the scehdule is defined as $C_{max} := max\{C_1, \dots, C_n\}$, and the total completion time is defined as $\sum_{j=1}^{n} C_j$. The latter is also referred as the min-sum objective in this work.

The *span* of a job j, $\lambda_j = r_j - l_j + 1$, is the number of machines on the path of j. The *length* of job j, $L_j = p_j \cdot \lambda_j$, is the minimum total time that job j needs to be completed. We say job j is *delayed* on machine i if the start time of j on i is strictly greater than its arrival time. The completion time C_j of job j is then equal to its *length* plus the total amount of time it has been delayed. Let C be the largest *congestion* over all machines (the maximum total running time of jobs that use machine i over all machines) and D be the maximum *length* over all jobs. Then clearly, both C and D are lower bounds for the makespan of the optimal schedule. This trivial lower bound has been used in many earlier works in design of algorithms and proving lower bounds for various scheduling problems.

1.1 Related work

One of the most general version of scheduling problem is the job shop scheduling with unrelated machines. The first polynomial time approximation algorithm for this problem is an $O(\frac{\log^2(m\lambda_{max})}{\log\log(m\lambda_{max})})$-approximation ($\lambda_{max}$ is the maximum span) for the makespan objective, given by [17]. Later, [6] improves the result by a $O(\log\log(m\lambda_{max}))$ fator, this is also the best known result for this problem. If the amount of time that every job takes to be processed on any machine is the same, then we get the packet routing problem when machines are edges of a graph. Leighton et al. [12, 13] show that there always exist a schedule of length $O(lb)$, where $lb = \max\{C, D\}$ is the trivial congestion/dilation lower bound for makespan objective. Later, the authors in [8] present a constructive algorithm that finds a schedule of length at most $8.84(C + D)$. However, the algorithms for the unit-processing time case seem hard to be adapted to the general processing time case and the approximability of non-unit processing time for jobs is still open even for special cases where the network of machines form a simple structure such as a tree or even a path.

Li et al. [14] show that if there is an α-approximation w.r.t. the lower bound $lb = \max\{C, D\}$ for the makespan objective, then there is a $2e\alpha$-approximation algorithm for the min-sum objective. This provides a framework of converting the makespan objective to the min-sum objective without affecting the approximation ratio asymptotically. Acyclic job shop is a special case of the job shop problem, where each job can have at most one operation on each machine. Feige et al. [3] give an $O(\log lb \log\log lb)$-approximation for the makespen objective of this problem. They also show the upper bound is nearly tight by proving the existence of instances of shortest makespan $\Omega(\frac{lb \log lb}{\log\log lb})$ even when machines are identical. The GPS problem considered in this paper is a special case of the acyclic job shop, where the machines are identical and form a path.

The best known result for the GPS problem is due to [4], they present an $O(\min\{\log n\lambda_{max}, \log p_{max}\})$-approximation for (the more general problem of) acyclic job shop with identical machines, under both the makespan and min-sum objective. However, many special cases of GPS problem can actually be solved exactly in polynomial time or have an $O(1)$-approximation algorithms. For example, if the network of the machines form a rooted tree and all the job paths have to go through the root of the tree, then the problem becomes the *junction tree* problem studied in [4], for which they present a 4-approximation for makespan and $8e$-approximation for min-sum. For the special case of GPS where all jobs have the same processing time, [10, 1] show that the greedy furthest-to-go gives the optimal makespan. Conversely, [1] shows shortest-to-go gives optimal min-sum for unit-processing time case. Moreover, authors of [9] show that furthest-to-go algorithm computes the optimal makespan on non-nested instances for general processing times; non-nested means span of no job is completely within span of another. If all jobs need to be processed on all (identical) machines from left to right, then the problem becomes the proportionate flow shop. It is straitforward that any fixed priority rule would give optimal solution for the makespan objective; for the (weighted) min-sum objective, [16] gives an exact algorithm that runs in $O(n^2)$ time. Bi-directional version of the problem, where there are jobs moving from left to right and right to left , can be dealt with by interleaving the uni-directional algorithms

If $m = O(1)$, then many scheduling problems admit better approximation ratios. For example, [15] gives a PTAS for the open shop makespan minimization problem, [17] gives a $(2 + \epsilon)$-approximation algorithm for job shop, and [7] gives a PTAS for the flow shop. Note that all the results discussed are based on the fact that m is fixed. Hall [7] introduces the notion of *outline scheme*, which is used in a couple of our algorithms in a fundamental way.

The scheduling problems where networks of machines have other specific structures have been the subjects of many researches. For example, when the machines form a grid, [11] shows that by applying the furthest-to-go algorithm vertically and horizontally one can get a 3-approximation for the makespan for unit processing time case. Same approximation ratio applies to the rooted tree network, where jobs can either go vertically upward, vertically downward, or upward-downward. When the network of machines is a star and jobs start/end at leaves, [4] gives a 1.796-approximation for the min-sum objective, and a 7.279-approximation for the general processing time case.

1.2 Our Results

We study GPS with both makespan and min-sum objectives and present several approximation algorithms and hardness results.

▶ **Theorem 2.** *There is a PTAS for GPS with makespan objective when $m = O(\frac{\log^{1/6} n}{\log \log n})$.*

This result actually holds even on unrelated machines setting and therefore, generalizes the result of Hall [7] for the classical problem of flow shop. We use this result as a subroutine to prove the following for general values of m:

▶ **Theorem 3.** *For GPS with makespan objective there is an $O(\frac{\log m}{\log \log m})$-approximation.*

This improves the $O(\min\{\log n\lambda_{max}, \log p_{max}\})$-approximation of [4]. We obtain similar results for the min-sum objective. First we introduce a variant of the GPS problem, called the *segmented* GPS problem and we give a PTAS for it for when $m = O(\log^{1/6} n/ \log \log n)$. We then use this to prove the following:

▶ **Theorem 4.** *There is a PTAS for GPS with min-sum objective when $m = O(\frac{\log^{1/6} n}{\log \log n})$.*

▶ **Theorem 5.** *For GPS with min-sum objective there is an $O(\frac{\log m}{\log \log m})$-approximation.*

This improves the result of [4]. Finally, we show GPS is NP-complete under both objectives.

▶ **Theorem 6.** *The* Generalized Path Scheduling *problem (under both makespan and min-sum objective) is NP-complete.*

Proofs of Theorems 5 and 6, and most of Theorem 4 appear in the full version of this paper.

2 A PTAS for the makespan objective when m is sub-logarithmic

In this subsection we prove Theorem 2. For this theorem, we assume we have unrelated machine setting (while our other results work with identical machine setting), hence Theorem 2 generalize the result of Hall [7] for the classic flow shop problem. The algorithm is built based on *outline scheme* and linear program. Similar techniques have been used in [7, 15] to design PTAS for the flow shop and open shop problems when $m = O(1)$. The general framework is to break the jobs (based on their processing times) into *large* and *small* jobs. It can be shown that the number of large jobs cannot be too large and we can guess (enumerate) their schedule on the machines with good accuracy. For small jobs we find a good schedule using a Linear Programming (LP) relaxation and rounding with small error.

2.1 The outline scheme

Since we are assuming we have unrelated machines, we use p_{ij} to denote processing time of job j on machine i.

▶ **Definition 7.** *An outline scheme partitions all feasible solutions into classes (outlines), such that solutions that get grouped together share some common characteristics.*

The outline scheme should suggest a natural way to obtain a good schedule. Our goal is to show that: ① The number of *outlines* is polynomially bounded. ② For each *outline*, we can generate a schedule such that the makespan of the schedule is approximately $(1 + \epsilon)$ as good as the optimal schedule in this *outline*. Since the optimal schedule must be contained in one of those *outlines*, by enumerating all of them, we are guaranteed to find a nearly good schedule. Suppose we have an upper bound T on the length of the optimal schedule T^*. Such an upper bound can be obtained by using a naive algorithm that simply processes operations starting from M_1 and move to the next machine if all operations on the current machine are finished. Therefore $T \leq mT^*$ is always a valid upper bound. Then we partition

$$
\begin{aligned}
\sum_{t_1,t_2,\dots,t_{\lambda_j}} x_{j,(t_1,t_2,\dots,t_{\lambda_j})} &= 1, && j=1,\dots,n', \\
\sum_{\{j|M_1\in P_j\}} p_{1j}x_{j,(\dots,k,\dots)} &\le \alpha_1^k, && k=1,\dots,\kappa, \\
\sum_{\{j|M_2\in P_j\}} p_{2j}x_{j,(\dots,k,\dots)} &\le \alpha_2^k, && k=1,\dots,\kappa, \\
&\dots \\
\sum_{\{j|M_m\in P_j\}} p_{mj}x_{j,(\dots,k,\dots)} &\le \alpha_m^k, && k=1,\dots,\kappa, \\
x &\ge 0 && .
\end{aligned}
$$

Figure 1 The LP to assign small jobs. Recall that p_{ij} is the processing time of job j on machine i, and P_j is the path of machines for job j.

the time line from 0 to T into κ intervals of size $\delta = \frac{T}{\kappa}$, and we refer to interval $[(k-1)\delta, k\delta)$ as the k-th δ-interval, $1 \le k \le \kappa$. Values of δ and κ are to be determined. Also, we classify the jobs into *big* and *small* jobs. The *big* jobs are those with maximum processing time (over their span of machines) at least γ, and *small* jobs are those with maximum processing time $< \gamma$. The value of γ will be specified later. Then we are ready to formally define the *outline scheme*. Each *outline* consists of:

- The δ-interval in which each operation of a *big* job begins.
- For each machine and δ-interval, the approximate (rounded up to the nearest multiple of γ) amount of time allocated to the operations of *small* jobs that begin in that δ-interval.

Therefore, the *outline* specifies which δ-interval each operation of each *big* job should begin in, and how much small-jobs-time is allocated for each δ-interval on each machine. The reason that we label jobs as *big* and *small* is because we cannot afford to guess too much detail on every job. Instead, for the *small* jobs, whose order of scheduling do not impact the overall makespan significantly, we can schedule them approximately by using an LP. How many *outlines* do we need to guess? Suppose the number of *big* jobs is L, then the number of possible assignments of *big*-job operations to δ-intervals is at most κ^{mL}. And observe that the number of possible assignments of small-jobs-time to intervals is at most $(\frac{\delta}{\gamma}+1)^{m\kappa}$.

Hence, the number of outlines is bounded by: $\kappa^{mL}(\delta/\gamma+1)^{m\kappa}$. We will choose the parameters in such a way that the number of possible outlines is bounded by a polynomial. So we can enumerate over all possible outlines (each of which tells us how the operations of the big jobs are to be ordered and roughly how much time we are to allocate to small jobs in each interval). When we find a solution we allow each interval to be expanded slightly; i.e. the time we spend to perform the operations of the jobs for each interval is slightly bigger than what the interval size is. This small over usage over all intervals results in small increase in the total makespan of the final schedule.

For a given outline, we introduce a Linear Program to determine the assignment of small-job operations to δ-intervals. Let $J_1, J_2, \dots, J_{n'}$ be the small jobs, and job J_i is to be processed on machines $M_{i_1}, \dots, M_{i_{\lambda_i}}$ (recall $\lambda_i \le m$ is the span of job J_i) in the specified order. Then we construct an LP with the following variables:

$$x_{j,(t_1,t_2,\dots,t_{\lambda_j})}, j=1,\dots,n', 1\le t_1\le t_2\le\dots\le t_{\lambda_j}\le\kappa,$$

where $x_{j,(t_1,t_2,\ldots,t_{\lambda_j})} = 1$ means that job J_j is assigned to δ-interval t_1 on machine M_{j_1}, t_2 on machine M_{j_2}, and so on. We use $\alpha_1^k, \alpha_2^k, \ldots, \alpha_m^k$ to denote the amount of time (for *small* jobs) assigned (by *outline*) to the k-th δ-interval on machines $M_1, M_2, \ldots, M_m$, respectively. We want to find a basic feasible solution against the constraints in Fig. 1. The first and last constraints ensure that the operations of all *small* jobs are assigned to some δ-intervals, and all constraints in the middle ensure that the small-job-time in the solution in each interval on each machine is no more than the value described by the outline. Observe that the LP has $n' + m\kappa$ constraints and at most $n'\kappa^m$ variables. A basic feasible solution (bfs) of this LP is guaranteed to have at most $n' + m\kappa$ positive variables. Also, each job must have at least one positive variable associated with it, this is because of the first constraint of the LP. Thus, a job that receives fractional assignment must have at least one more positive variable. Combining with the fact that the bfs has at most $n' + m\kappa$ positive variables, we know that such a solution can have at most $m\kappa$ jobs that actually receive fractional assignments and the remaining small jobs will have unique integral assignment to δ-intervals. Let's just ignore the small jobs that received fractional assignments. They will be appended to the end of the schedule with a cost of at most $(m\kappa + m - 1)\gamma$. For the remaining jobs (*big* jobs + *small* jobs with integral assignments), we describe a two-step algorithm to construct a schedule based on their assignments to δ-intervals.

In the first step, we 'greedily' schedule each machine independently. For a machine M_i, we order the operations assigned to each δ-interval such that the longest operation is the last (for analysis purposes). More precisely, let I be the indices k such that there are some operations assigned to the kth δ-interval in the first step schedule. Then we schedule the operations in the order of their indices in I where operations in kth δ-interval start at time σ_k and end at time τ_k, where $\sigma_k = max\{(k-1)(\delta+\gamma), max_{1\le h\le k-1, h\in I}\{\tau_h\}\}$, τ_k becomes well-defined once we defined σ_k. Another way to view the first step schedule is that: for machine M_i, all operations assigned in the first δ-interval get scheduled first as a block with no idle time in between, followed by the operations in the second δ-interval, and so on. Within each block, we schedule the largest operation the last. Therefore, jobs in each block do not overlap, and they are not scheduled before the specified starting time.

Let Σ be the optimal schedule in a fixed outline, and let $\tilde{T}$ be its length. We focus on a specific machine M_i. Let s_k and (t_k) denote the start time and (end time) of the first and (last) operations during the kth δ-interval in Σ. Then using a simple induction (proof appears in the full version of this paper) we can show:

▶ **Lemma 8.** *For all k, $\sigma_k \le s_k + (k-1)\gamma$, and $\tau_k < t_k + k\gamma$.*

▶ **Corollary 9.** *The makespan of the first step schedule is at most $\tilde{T} + \kappa\gamma$.*

However, notice that the first-step schedule is very likely an infeasible schedule, because we only focus on each machine individually, so the operations of a job might get processed on different machines at the same time (overlaps). The second step of the algorithm is to remove the potential overlaps of operations by delaying the operations on M_i by $2(i-1)(\delta+\gamma)$ units of time, for $i = 2, \ldots, m$. We will eventually show that the schedule after injecting delays on every machine will be feasible, but before that we need to prove the following lemma first:

▶ **Lemma 10.** *Consider operations inside an arbitrary kth δ-interval on a arbitrary machine in the first step schedule. (1) Each large operation starts processing during $[(k-1)(\delta+\gamma), k(\delta+\gamma))$. (2) And each small operation starts and finishes processing during $[(k-1)(\delta+\gamma), k(\delta+\gamma)+\gamma)$.*

Proof. It is clear that all these operations start at or after $(k-1)(\delta+\gamma)$. Then it remains to show that they don't start (end) too late. We consider the following two cases. First, suppose there exists a large operation. Observe that all large operations are scheduled in the same δ-interval in Σ (recall Σ is the optimal schedule in the *outline*) as well. Let O_j be the operation that was scheduled the last in our algorithm (it is the largest), then it suffices to show O_j starts before time $k(\delta+\gamma)$). From Lemma 8, we know $\tau_k < t_k + k\gamma$, also there is some operation $O_{j'}(p_j \geq p_{j'})$ is scheduled the last and completes at t_k in Σ. Also, $O_{j'}$ starts before time $k\delta$. Therefore, the last operation of this interval starts at time $\tau_k - p_j < t_k + k\gamma - p_{j'} < k(\delta+\gamma)$. The other case is when all operations are small ($< \gamma$). In this case , $t_k < k\delta + \gamma$, so by Lemma 8: $\tau_k < t_k + k\gamma < k(\delta+\gamma)+\gamma$. ◀

▶ **Lemma 11.** *After delaying the operations on M_i by $2(i-1)(\delta+\gamma)$ units, for $i = 2, \ldots, m$, the resulting schedule is feasible.*

Proof. The schedule we obtained from first step is conflict-free in each interval, but it is still likely infeasible because there might be a job starting on a machine before its previous operation finishes on the previous machine (the job starts before it becomes available). In step two, we delay operations in M_2 by $2(\delta+\gamma)$, jobs in M_3 by $4(\delta+\gamma)$, and so on. So the makespan of the schedule increases by at most $2(m-1)(\delta+\gamma)$. And we show that the schedule after injecting delays is feasible. Consider an arbitrary job j, and two consecutive operations of j on machines M_i and M_{i+1}, call them $O_{j,i}$, and $O_{j,i+1}$. It suffices to prove that these two operations are scheduled in order and do not overlap.

First consider the case when j is a *big* job. Suppose $O_{j,i}$ is assigned to the kth δ-interval and $O_{j,i+1}$ is assigned to the lth δ-interval ($l \geq k$). By Lemma 10, the difference of their starting time is at least $(l-k-1)(\delta+\gamma)$, i.e. in the worst case $O_{j,i+1}$ starts on M_{i+1} $(\delta+\gamma)$ units before $O_{j,i}$ starts on M_i in the first step schedule. Note that operations on M_{i+1} are delayed by $2(\delta+\gamma)$ more units relative to operations on M_i in step two, so once the delays have been injected, $O_{j,i}$ and $O_{j,i+1}$ will be scheduled in order and do not overlap.

Another case is when j is a small job, and we still use $O_{j,i}$ and $O_{j,i+1}$ to denote the two consecutive operations of j. And suppose $O_{j,i}$ is assigned to kth δ-interval and $O_{j,i+1}$ is assigned to the lth δ-interval ($l \geq k$). Again by Lemma 10, after the delays have been injected, $O_{j,i}$ will complete before $(k+2(i-1))(\delta+\gamma)+\gamma$, and $O_{j,i+1}$ will start at or after time $(2i+k-1)(\delta+\gamma)$. Since $(2i+k-1)(\delta+\gamma) > (k+2(i-1))(\delta+\gamma)+\gamma$, $O_{j,i}$ and $O_{j,i+1}$ will be scheduled in order and do not overlap. ◀

Combining previous lemmas, we obtain the following theorem:

▶ **Theorem 12.** *For a given δ, and γ and an outline with an associated optimal schedule of length $\tilde{T}$, we can generate a feasible schedule of length:*

$$\tilde{T} + \kappa\gamma + 2(m-1)(\gamma+\delta) + (m\kappa + m - 1)\gamma$$

Proof. The additive $\kappa\gamma$ follows from Corollary 9, second term $2(m-1)(\gamma+\delta)$ follows from Lemma 11, and $(m\kappa+m-1)\gamma$ comes from the fractional *small* jobs that get appended at the end. ◀

2.1.1 The PTAS

In this section, we show that the algorithm that we obtain from previous section is a PTAS for the GPS makespan minimization problem. Let $\delta = \frac{T\epsilon}{u}$, and $\gamma = \frac{T\epsilon^2}{uv}$. The value of u and v will be specified later. So the additive error from Theorem 12 becomes:

$$\begin{aligned}
Error &= \kappa\gamma + 2(m-1)(\gamma+\delta) + (m\kappa + m - 1)\gamma \\
&= \frac{u}{\epsilon}\cdot\frac{T\epsilon^2}{uv} + 2(m-1)\left(\frac{T\epsilon(v+\epsilon)}{uv}\right) + \left(\frac{mu}{\epsilon} + m - 1\right)\frac{T\epsilon^2}{uv} \\
&= \left[\frac{3m\epsilon + 2mv + (m+1)u - 3\epsilon - 2v}{uv}\right]\epsilon T
\end{aligned}$$

Moreover, recall that T is the upper bound of $\tilde{T}$, suppose $T = \beta\tilde{T}$, such β is at most m. Let L be the number of large jobs, then L is at most $\frac{m\tilde{T}}{\gamma} = \frac{muv}{\beta\epsilon^2}$, therefore the total possible number of assignments of large operations to δ-intervals is at most κ^{mL}. The number of small-job-time that we assign to each δ-interval is $\frac{\delta}{\gamma} + 1 = \frac{v}{\epsilon} + 1$ (because we round it up to multiple of γ), so the number of possible assignments of small-job-time each machine during each interval is at most $(\frac{v}{\epsilon}+1)^{m\kappa}$. Therefore, the total number of outlines is at most:

$$\left(\frac{u}{\epsilon}\right)^{m^2uv/\epsilon^2}\left(\frac{v}{\epsilon}+1\right)^{mu/\epsilon}$$

Assuming $u = 4(m-1)\beta$ and $v = 2(m+1)\beta + (\frac{3}{2})\epsilon$, then the additive error becomes:

$$\begin{aligned}
Error &= \left[\frac{3m\epsilon + 4m(m+1)\beta + 3m\epsilon + 4(m+1)(m-1)\beta - 3\epsilon - 4(m+1)\beta - 3\epsilon}{8(m+1)(m-1)\beta^2 + 6(m-1)\beta\epsilon}\right]\cdot\epsilon\beta\tilde{T} \\
&= \left[\frac{8(m+1)(m-1)\beta + 6(m-1)\epsilon}{8(m+1)(m-1)\beta^2 + 6(m-1)\epsilon\beta}\right]\cdot\epsilon\beta\tilde{T} = \epsilon\tilde{T}
\end{aligned}$$

Therefore, we can guarantee the additive error is at most $\epsilon\tilde{T}$.

Runtime. The runtime is given by the number of *outlines* that we need to consider multiplied by the time needed to solve an individual *outline*. First note that the number of *outlines* is $(m/\epsilon)^{O(m^6/\epsilon^2)}$. For each *outline*, we need to find a basic feasible solution for the associated LP; recall that the LP has $n + \frac{mu}{\epsilon}$ constraints and at most $\frac{nu^m}{\epsilon^m}$ variables. By using the LP solver from [20], we can solve the LP in time $O(N^{3.5})$, where N is the input size. Therefore the total runtime is $O(N^{3.5}(m/\epsilon)^{O(m^6/\epsilon^2)})$. Suppose $m = O(\frac{\log^{1/6} n}{\log\log n})$, the total runtime becomes:

$$O(N^{3.5}m^{(m^6)}) = O\left(N^{3.5}\left(\frac{\log^{1/6} n}{\log\log n}\right)^{\log n/\log^6\log n}\right) = O(N^{3.5}n).$$

Therefore, the total runtime is polynomial, which completes the proof of Theorem 2.

3 An $O(\frac{\log m}{\log\log m})$-approximation for makespan objective

In this Section we prove Theorem 3. Consider the special case of the GPS problem (with identical machines) where there are h machines, called *terminal* machines, such that any job j has to start/end at one of the h machines. We show how an ρ-approximation for this special case can be used to derive an $O(\rho\log_h m)$-approximation for general case of GPS.

For a given instance of GPS, we select h machines (including the first and last machines) that partition the path of machines into $h-1$ segments of equal sizes (or sizes that differ by at most 1), call these h machines level 1. For all the jobs whose span crosses these h machines (i.e. uses machines in more than two segments), we group them into group G_1. So all the jobs in $J - G_1$ have their span entirely within one segment and those that fall into different segments, can be scheduled independently (as their paths do not overlap). For each segment, we again select h machines (to partition that segment into equal segments) and all the jobs in $J - G_1$ that pass any one of the second level terminal machines (h^2 many) form group G_2. And we do this recursively. Eventually, we have partitioned the jobs into $O(\log_h m)$ groups. Also, by losing a constant factor, we can assume that jobs among a group have to start/finish at one of the terminal machines.[1] Suppose we have a ρ-approximation for a single group of jobs, then if we schedule all groups sequentially, we obtain an $O(\rho \log_h m)$-approximation for the general case. We will show below that we can set $h = O(\log^{1/6} m/ \log\log m)$ and will have $\rho = O(1)$.

3.1 Instances with h terminal machines

In this section, we show how one can extend the idea of the PTAS in Section 2 to solve the instances with h terminal machines. Similarly, suppose we have an upper bound T on the *makespan* of the optimal schedule, and we partition the time line from 0 to T into κ intervals of size $\delta = \frac{T}{\kappa}$. The definition of a job being *big* or *small* is slightly different. Suppose the h terminal machines partition the machines into $h-1$ equal-size segments (except the last one). A job j is *big* if the time it takes to travel a segment is $\geq \gamma$ (i,e, $p_j \times$ *segment size* $\geq \gamma$). Otherwise, we say the job is *small*. Each *outline* should specify:

- The δ interval in which a *big* job starts running on a terminal machine.
- For each terminal machine and δ-interval, how much time is allocated to small jobs that begins in that δ-interval, rounded up to the nearest multiple of γ.

Suppose the number of *big* jobs is L, the number of guesses of the starting time interval of all *big* jobs is at most κ^{hL}; also, the number of possible assignments of small-job-time to δ-intervals is at most $(\delta/\gamma+1)^{h\kappa}$. So the number of outlines is at most: $k^{hL}(\delta/\gamma+1)^{h\kappa}$. For the small jobs, we again construct an LP as in Section 2 and find a basic feasible solution of it. Then we have at most $h\kappa$ jobs that actually receive fractional assignments, we can ignore them for now and append them at the end of the schedule with a cost at most $(h\kappa+h-1)\gamma$.

For all *big* jobs and *small* jobs with integral assignments, we schedule them according to their assignments to δ-intervals in two steps. In the first step, we schedule each segment independently. For a fixed segment, let M_i be the first machine (a terminal machine) of this segment. We order the jobs assigned to each δ-interval according to their processing times and send them based on *faster first*. Let σ_k be the time when the first job in the kth δ-interval begins on the first machine of the segment, and let τ_k be the time when the last job in the kth δ-interval finishes on the last machine of the segment.

Let Σ be the optimal schedule in the *outline* that we are focusing on, say the makespan of Σ is $\tilde{T}$. Similarly, we define s_k and (t_k) to be the start time (end time) of the first (last) job during the kth δ-interval of Σ on the same segment. Then we show the following (proof appears in the full version of this paper)

▶ **Lemma 13.** *For all k, $\sigma_k \leq s_k + (k-1)\delta$, and $\tau_k < t_k + k\gamma$.*

[1] This is because for instances where there is a machine that is used by all jobs it is a special case of the *junction tree* problem studied in [4]. Therefore we can use their two-stage algorithm and in $\leq 2OPT$ time send all the jobs to their first terminal machines, and once all the jobs reach their last terminal machines on their paths, spend another $\leq 2OPT$ time to deliver them to their final destinations.

Then we can conclude the *makespan* of the schedule obtained from the first step is at most $\tilde{T} + \kappa\gamma$. However, this schedule is likely to be infeasible because we schedule each segment independently without caring about their consistency. The second step is to inject delays to jobs so that the jobs in the resulting schedule are processed in order. We delay operations on the second terminal machine by $2(\delta + \gamma)$, delay operations on the third terminal machine by $4(\delta + \gamma)$, and so on. So eventually, if we consider two adjacent terminal machines, operations on the later one are delayed by $2(\delta + \gamma)$ units relative to the previous terminal machine. And we need to show:

▶ **Lemma 14.** *After delaying the operations on the ith terminal machine M_{h_i} by $2(i-1)(\delta+\gamma)$ units of time, for $i = 2, \dots, h$. The resulting schedule is feasible.*

Proof. Consider an arbitrary job j and its operations traveling two adjacent terminal machines M_{h_i} and $M_{h_{i+1}}$. Observe that all operations of job j in the segment starting with machine M_{h_i} must be scheduled in order because we schedule them based on *faster first*. So it remains to show that after injecting the delays, j doesn't start on $M_{h_{i+1}}$ before all previous operations are finished. The rest of proof is analogous to the proof of Lemma 11. ◀

Therefore, the final schedule is of length at most $\tilde{T} + \kappa\gamma + 2(h-1)(\delta+\gamma) + (h\kappa + h - 1)\gamma$. For sufficiently small $\delta = O(\frac{T\epsilon}{h})$ and $\gamma = O(\frac{T\epsilon^2}{h^2})$, the additive error becomes $\epsilon\tilde{T}$. Moreover, the total number of *outlines* is $O(h^{(h^6)})$. Suppose h is sub-logarithmic, say $h = \frac{\log^{1/6} m}{\log\log m}$, then runtime becomes polynomially bounded. This implies a $O(\rho \log_h m) = O(\frac{\log m}{\log\log m})$-approximation for the general problem, which completes the proof of Theorem 3.

4 Approximations for min-sum objective

In this section, we study the approximability of the *min-sum* objective and prove Theorem 4. Proof of Theorem 5 uses Theorem 4 and appears in the full version of this paper. The ideas of designing approximation algorithms for the min-sum objective using algorithms for the min-max (makespan) variants have been used extensively for various problems such as scheduling and vehicle routing problems (to name a few see e.g. [2, 14, 18, 4]). Here we borrow ideas from [18], which designs a PTAS for minimum-latency traveling repairman problem on Euclidean metrics by reducing it to a variant of min-max version of it. This technique is used to design algorithms for many other problems, see [5, 19] for an example. First we introduce a variant of GPS called segmented GPS and present a PTAS for it when m is sub-logarithmic using ideas of Theorem 2. Then in Section 4.1, we use it as a subroutine to design a PTAS for the *min-sum* objective GPS problem for sub-logarithmic m. Finally, in Section 4.2, we present an $O(\frac{\log m}{\log\log m})$-approximation for *min-sum* GPS with general m.

4.1 A PTAS for Min-Sum GPS when m is sub-logarithmic

In this section we prove Theorem 4 (all missing proofs appear in the full version of this paper). In order to do so we first define an interesting variant of the GPS problem called the *segmented* GPS as follows.

▶ **Definition 15** (*segmented* GPS)**.** *An instance of segmented GPS is given by a set of m identical machines that form a path, and also a set of n jobs each needs to be processed on a sub-path. Also, for some constant π, given bounds $B_1 \le B_2 \le \dots \le B_\pi$ such that $B_i/B_{i-1} = \eta$ where η is a constant, and given numbers $n_1 \le n_2 \le \dots \le n_\pi = n$. A feasible solution is a schedule such that at least n_i jobs are finished within the first B_i units of time* *for all $i \in \{1, \dots, \pi\}$, and the length of the schedule is at most B_π. We say an algorithm gives an α-approximation if for any feasible instance it finds a schedule that finishes at least n_i jobs within αB_i units of times, for all $1 \le i \le \pi$.*

We can prove the following similar to Theorem 2.

▶ **Theorem 16.** *There is a PTAS for segmented GPS when $m = O(\frac{\log^{1/6} n}{\log\log n})$.*

The algorithm adapts the idea of *outline scheme*. Intuitively, we again partition the time line from 0 to B_π into polynomially many δ-intervals, and we use the notion of *big* and *small* to classify jobs so that we can afford to fully guess the assignment of *big* jobs to δ-intervals. For *small* jobs, we *guess* approximately the amount of time that is allocated for them on each δ-interval and each machine, and we then assign *small* jobs by an LP.

However, we define δ w.r.t. B_1 instead of B_π. This gives us better precision so that the additive error at the end depends on B_1, so is relatively small. Also, at the same time, the number of δ intervals doesn't blow up, the number is at most $\eta^{\pi+1}$ (which is a constant since both η, π are constants) times what we used to have. Moreover, for the same reason, we define γ w.r.t. B_1. The number of large jobs L is also $\eta^{\pi+1}$ (which is a constant) times what we used to have.

More precisely, let $\delta = \frac{B_1 \epsilon}{u}$ and $\gamma = \frac{B_1 \epsilon^2}{uv}$, as before, $u = O(m)$ and $v = O(m)$ are functions of m to be specified later. The number of δ-intervals is $\kappa \le \frac{\eta^{\pi+1} B_1}{\delta} = \frac{u\eta^{\pi+1}}{\epsilon}$. An *outline* specifies the δ-interval in which an operation of a *big* jobs begins in, and amount of time that is allocated to *small* job in each interval on each machine, rounded to nearest multiple of γ. Therefore, the total number of *outlines* is $\kappa^{mL}(\frac{\delta}{\gamma}+1)^{m\kappa} = O(m^{(m^6)})$, which is polynomially bounded as m is sub-logarithmic.

Assume we know the assignment of *big* jobs. As before, we process the operations assigned to each δ-intervals s.t. the longest operation is the last. Then, for each bound B_i, we know the number of large jobs n_i^l that are finished before B_i. Therefore, when we assign *small* jobs, we modify the LP in Figure. 1 by adding π extra constraints to ensure that x values that fall in the first B_i units of time is at least $n_i^s = n_i - n_i^l$ (see Figure. 2 for the full LP).

Such LP has $n' + m\kappa + \pi$ constraints and at most $n'\kappa^m$ variables (recall n' is the number of small jobs). A basic feasible solution of this LP is guaranteed to have at most $m\kappa + \pi$ small jobs that actually receive fractional assignments and the remaining small jobs will have unique integral assignment to δ-intervals. We can simply ignore the fractional small jobs for now, because we can append them all at the end of B_1 with a cost of at most $(m\kappa + \pi + m - 1)\gamma$ (which is at most ϵB_1, based on the discussion in Section 2). Call this schedule the first-step schedule, then it finishes at least n_i jobs before time B_i, for $i = 1, \dots, \pi$, as wanted. But it may not be feasible. In order to turn it into a feasible schedule, we need to inject delays to machines.

▶ **Lemma 17.** *After delaying the operations on machine M_i by $2(i-1)(\delta+\gamma)$ units, for $i = 2, \dots, m$. The resulting schedule becomes feasible. And the delays only stretch the schedule by a factor of $(1+\epsilon)$.*

Proof. The proof is analogous to the proof of Lemma 8 and Theorem 12. ◀

Theorem 16 follows immediately from the above discussion. This theorem combined with the following implies Theorem 4:

▶ **Theorem 18.** *If there is a polynomial time α-approximation algorithm for the segmented GPS problem, then there is a polynomial time $(1+\epsilon)\alpha$-approximation algorithm for the GPS min-sum minimization problem.*

$$\sum_{t_1,t_2,\dots,t_{\lambda_j}} x_{j,(t_1,t_2,\dots,t_{\lambda_j})} = 1, j = 1,\dots,n',$$
$$\sum_{\{j|M_1\in P_j\}} p_j x_{j,(\dots,k,\dots)} \le \alpha_1^k, k = 1,\dots,\kappa,$$
$$\sum_{\{j|M_2\in P_j\}} p_j x_{j,(\dots,k,\dots)} \le \alpha_2^k, k = 1,\dots,\kappa,$$
$$\vdots$$
$$\sum_{\{j|M_m\in P_j\}} p_j x_{j,(\dots,k,\dots)} \le \alpha_m^k, k = 1,\dots,\kappa,$$
$$\sum_{\{j|t_{\lambda_j}\cdot\delta\le B_i\}} x_{j,(\dots,t_{\lambda_j})} \ge n_i^s, i = 1,\dots,\pi,$$
$$x \ge 0.$$

Figure 2 the modified LP. π new constraints are added to ensure that at least $n_i^s = n_i - n_i^l$ small jobs are finished before time B_i.

Proof of this theorem is built upon ideas of [18] for minimum latency traveling repairman problem on Euclidean metrics. With a $(1+\epsilon)$-factor loss, we may assume that the makespan of the optimal schedule is polynomially bounded in m, n. [2] The reduction is as follows. Consider the time points $t_1, t_2, \dots, t_\Gamma$, where $t_1 = 1$ and $t_i/t_{i-1} = (1+\epsilon)^\pi$, for some constant π that only depends on ϵ ($\pi = O(1/\epsilon^2)$), and we can assume $\Gamma = O(\log(mn))$. The part of schedule between t_i and t_{i+1} is called the ith *subschedule*. We call a schedule is *well-structured* if each *subschedule* processes a subset of jobs completely. That is, if a job j starts processing at or after time t_i, it has to be finished on all its span before t_{i+1}. We first show that we can reduce the solution space to only the *well-structured* schedules by losing an ϵ-factor. This allows us to deal with each *subschedule* independently. Moreover, the time frame between t_i and t_{i+1} can be further partitioned into π sub-intervals such that the ratio of the end time and start time of each sub-interval is $(1+\epsilon)$. Therefore, each *subschedule* can be viewed as an instance of the *segmented GPS* problem. However, we cannot afford to guess the subset of jobs to be processed on every *subschedule*, but we show that, for large enough π, in the ith *subschedule* we can simply re-do all the jobs that have been processed in the previous *subschedules*. As a result, we don't need to know the set of jobs to be processed on each *subschedule*, instead, we use Dynamic Programming to enumerate the number of job to be processed, which can be done in polynomial time. The first step is the following lemma (proof appears in the full version of this paper).

▶ **Lemma 19.** *There is a $(1+\epsilon)$-approximate well-structured schedule OPT'.*

So using Lemma 19 we can focus on *well-struuctured* solutions. The proof of this lemma shows that solution OPT' is ① *well-structured*; ② each *subschedule* processes all jobs that appear in previous *subschedules*. Therefore, let D_j be the completion time of

[2] This is a fairly standard trick. If p_{max}, p_{min} are the largest and smallest processing times (respectively) one can assume that $p_{min} \ge \epsilon p_{max}/(mn)$, otherwise all jobs smaller than $\epsilon p_{max}/(mn)$ can be removed, then they can be added to any schedule of the rest of the jobs right before a job of size p_{max} and this will increase the total completion time of the schedule by at most a $1+\epsilon$ factor. With this assumption we can scale processing times so that $p_{min} = 1$ and hence $p_{max} \le mn/\epsilon$.

jth job on the first *subschedule* that processes at least j jobs in OPT', we know that $\sum_{j=1}^{n} D_j = \sum_{j\in J} C'_j \leq (1+\epsilon)opt$. Therefore, in order to find such an OPT', we can search for a solution among all schedules satisfying ① and ② that minimizes $\sum_{j=1}^{n} D_j$. We show this can be done by a DP that runs in polynomial time if we have an algorithm for the following subproblem.

▶ **Definition 20** (The subproblem). *An instance of the subproblem is given by $i \in \{1,\ldots,\Gamma\}$ and integers $n' \leq n'' \in \{0,1,\ldots,n\}$. A solution is a schedule that starts at time t_i and finishes before time t_{i+1} that processes exactly n'' jobs. The goal is to find a schedule that minimizes the total completion time of jobs $n'+1,\ldots,n''$. For any feasible instance (i,n',n''), let $\textsc{Sub}_i(n',n'')$ denote its optimal value.*

We say an algorithm is (α,β)-approximation for the subproblem if for any feasible instance (i,n',n''), it finds a schedule that starts at time αt_i and finishes before time αt_{i+1}, and the total completion time of jobs $n'+1,\ldots,n''$ is at most $\alpha\beta\textsc{Sub}_i(n',n'')$.

▶ **Lemma 21.** *If there is an (α,β)-approximation for the subproblem, then there is an $\alpha\beta(1+\epsilon)$ approximation for the GPS min-sum minimization problem.*

▶ **Lemma 22.** *If there is an α-approximation for the segmented GPS problem, then there is an $(\alpha,1+\epsilon)$-approximation for the subproblem.*

Theorem 18 follows from Lemmas 19, 21, and 22. Combining with the $(1+\epsilon)$-approximation for the *segmented* GPS problem from Theorem 16, we obtain a PTAS for the GPS *min-sum* minimization problem and hence prove Theorem 4. The number of subproblems that we need to consider is $O(\Gamma n^2) = O(n^2 \log(mn))$, and for each subproblem, we enumerate $O(n^{\pi}) = O(n^{1/\epsilon^2})$ instances of *segmented* GPS.

4.2 An $O(\frac{\log m}{\log\log m})$-approximation for Min-sum GPS

As mentioned earlier, the framework of using a min-max solver as a blackbox to approximate a min-sum objective has been used in the past extensively. To apply that here we first define the following variant of the problem:

▶ **Definition 23** (Throughput Maximization Given Bound B). *Given an instance of GPS and a bound B, what is the maximum number of jobs q that can be finished before this bound? An α-approximation for this problem is an algorithm that finishes q jobs within time αB.*

For the ease of notation, we denote this problem as problem A. Then

▶ **Lemma 24.** *If there is an α-approximation for problem A, then there is an $O(\alpha)$-approximation for the min-sum objective.*

Proof. Given a black box that can approximate problem A within factor α, we can obtain an $O(\alpha)$-approximation for the *min-sum* objective as follows. Let S_j be the set of jobs that finish between time 2^j and 2^{j+1} in the optimal schedule (regarding min-sum), and let $n_j = |S_j|$. Therefore, by invoking the solver for problem A, for each j, we can find a maximum set of jobs Q_j with size q_j that can be scheduled within time $\alpha 2^{j+1}$.

Our solution to the min-sum objective is the following: for $j = 1,2\ldots$, schedule the jobs in Q_j as suggested by the solver of problem A. Note that a job might be scheduled multiple times in different Q_j's, the completion time of a job is the first time when it is completely scheduled. Consider the ith job that finishes in our schedule, say $i \in S_j$. Then the completion time of ith job in the optimal schedule is at least 2^j. Consider the set of jobs

Q_j and note the $q_j \geq i$. Therefore the completion time of the ith job in our schedule is at most $\alpha \sum_{k=1}^{j+1} 2^k \leq \alpha 2^{j+2}$. That is, the average completion time of our schedule is at most 4α times the value of the optimal solution. This completes the proof. ◀

So it is enough to get an $O(\frac{\log m}{\log\log m})$-approximation for problem A (for general m). The algorithm is similar to the one in Section 3.1, so we only provide a sketch here. First, we select $h = O(\frac{\log^{1/6} m}{\log\log m})$ terminal machines that partition the machines into $h-1$ equal size segments. Group the jobs that cross the terminal machines together and do it recursively, we obtain $\Delta = O(\log_h m) = O(\frac{\log m}{\log\log m})$ classes of jobs. Let OPT be an optimal schedule that completes $q = q_1 + \cdots + q_\Delta$ jobs before given bound B, where q is the maximum possible jobs that can be finished before time B and q_i is the number of jobs from class i. Consider the instance of problem A on a single class of jobs. Similarly we consider the segment between two terminal machines as the role of a single machine and define δ and γ accordingly. Then each instance can be viewed as a special case of the *segmented* GPS problem when $\pi = 1$, so the PTAS for the *segmented* GPS problem can be applied here. That is, given bound B and let q_i^* be the maximum number of jobs that can be finished before B if we only consider jobs in class i, we have an algorithm that finishes q_i^* jobs before time $(1+\epsilon)B$. Note that $q_i^* \geq q_i$. Therefore, if we apply the PTAS on every class of jobs to obtain the q_i^* many jobs from each class i and sequentially glue them together, then we get a schedule that finishes $q_1^* + \cdots + q_\Delta^* \geq q_1 + \cdots + q_\Delta = q$ jobs before time $(1+\epsilon)\Delta B$, which is a $\Delta = O(\frac{\log m}{\log\log m})$-approximation for problem A. Combining with the result from Lemma 24, we get an $O(\frac{\log m}{\log\log m})$-approximation for *min-sum* objective (for general m).

5 Conclusion

We have proposed two $O(\frac{\log m}{\log\log m})$-approximation algorithms for the GPS problem under the makespan and min-sum objectives, and a PTAS for when m is sub-logarithmic. The problem of getting an $O(1)$-approximation algorithm for the GPS is still open for both objectives. The *furthest-to-go* algorithm seems plausible as it gives the optimal solution for the non-nested instances [9] and we do not know any example showing that the congestion/dilation lower bound is violated by more than a small constant factor by the *furthest-to-go* algorithm. It is also worth pointing out that, for the makespan objective, if one can show every fixed priority gives $O(1)$-approximation for instances in which all jobs have the same end machine, then *furthest-to-go* gives $O(1)$-approximation for GPS. This is because for any machine M_i in a GPS instance, if J_i is the set of jobs that use M_i, then none of the jobs in J_i will be delayed by any job in $J - J_i$ before machine M_i. The priority among jobs in J_i is defined by their destinations. Therefore, if every fixed priority gives $O(1)$-approximation for instances with same end machine, then the *furthest-to-go* algorithm gives $O(1)$-approximation for the completion time of every machine, hence for the makespan of the schedule.

References

1 Antonios Antoniadis, Neal Barcelo, Daniel Cole, Kyle Fox, Benjamin Moseley, Michael Nugent, and Kirk Pruhs. Packet forwarding algorithms in a line network. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, volume 8392 of *Lecture Notes in Computer Science*, pages 610–621. Springer, 2014. doi:10.1007/978-3-642-54423-1_53.

2 Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 36–45. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238179.

3 Uriel Feige and Christian Scheideler. Improved bounds for acyclic job shop scheduling (extended abstract). In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 624–633. ACM, 1998. `doi:10.1145/276698.276878`.

4 Zachary Friggstad, Arnoosh Golestanian, Kamyar Khodamoradi, Christopher S. Martin, Mirmahdi Rahgoshay, Mohsen Rezapour, Mohammad R. Salavatipour, and Yifeng Zhang. Scheduling problems over a network of machines. *J. Sched.*, 22(2):239–253, 2019. `doi:10.1007/s10951-018-0591-z`.

5 Iftah Gamzu and Danny Segev. A polynomial-time approximation scheme for the airplane refueling problem. *J. Sched.*, 22(1):119–135, 2019. `doi:10.1007/s10951-018-0569-x`.

6 Leslie Ann Goldberg, Mike Paterson, Aravind Srinivasan, and Elizabeth Sweedyk. Better approximation guarantees for job-shop scheduling. *SIAM J. Discret. Math.*, 14(1):67–92, 2001. `doi:10.1137/S0895480199326104`.

7 Leslie A. Hall. Approximability of flow shop scheduling. *Math. Program.*, 82:175–190, 1998. `doi:10.1007/BF01585870`.

8 David G. Harris and Aravind Srinivasan. Constraint satisfaction, packet routing, and the lovasz local lemma. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 685–694. ACM, 2013. `doi:10.1145/2488608.2488696`.

9 Ronald Koch, Britta Peis, Martin Skutella, and Andreas Wiese. Real-time message routing and scheduling. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2009. `doi:10.1007/978-3-642-03685-9_17`.

10 Dariusz R. Kowalski, Eyal Nussbaum, Michael Segal, and Vitaly Milyeykovski. Scheduling problems in transportation networks of line topology. *Optimization Letters*, 8(2):777–799, 2014. `doi:10.1007/s11590-013-0613-x`.

11 Dariusz R. Kowalski, Zeev Nutov, and Michael Segal. Scheduling of vehicles in transportation networks. In Alexey V. Vinel, Rashid Mehmood, Marion Berbineau, Cristina Rico Garcia, Chung-Ming Huang, and Naveen Chilamkurti, editors, *Communication Technologies for Vehicles - 4th International Workshop, Nets4Cars/Nets4Trains 2012, Vilnius, Lithuania, April 25-27, 2012. Proceedings*, volume 7266 of *Lecture Notes in Computer Science*, pages 124–136. Springer, 2012. `doi:10.1007/978-3-642-29667-3_11`.

12 Frank Thomson Leighton, Bruce M. Maggs, and Satish Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994. `doi:10.1007/BF01215349`.

13 Frank Thomson Leighton, Bruce M. Maggs, and Andréa W. Richa. Fast algorithms for finding o(congestion + dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999. `doi:10.1007/s004930050061`.

14 Wenhua Li, Maurice Queyranne, Maxim Sviridenko, and Jinjiang Yuan. Approximation algorithms for shop scheduling problems with minsum objective: A correction. *J. Sched.*, 9(6):569–570, 2006. `doi:10.1007/s10951-006-8790-4`.

15 Sergey V. Sevastianov and Gerhard J. Woeginger. Makespan minimization in open shops: A polynomial time approximation scheme. *Math. Program.*, 82:191–198, 1998. `doi:10.1007/BF01585871`.

16 Natalia Shakhlevich, Han Hoogeveen, and Michael Pinedo. Minimizing total weighted completion time in a proportionate flow shop. *Journal of Scheduling*, 1998.

17 David B. Shmoys, Clifford Stein, and Joel Wein. Improved approximation algorithms for shop scheduling problems. *SIAM J. Comput.*, 23(3):617–632, 1994. `doi:10.1137/S009753979222676X`.

18 René Sitters. Polynomial time approximation schemes for the traveling repairman and other minimum latency problems. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 604–616. SIAM, 2014. `doi:10.1137/1.9781611973402.46`.

19 René Sitters and Liya Yang. A \$(2 + ϵ)\$-approximation for precedence constrained single machine scheduling with release dates and total weighted completion time objective. *CoRR*, abs/1706.07604, 2017. `arXiv:1706.07604`.

20 Pravin M. Vaidya. Speeding-up linear programming using fast matrix multiplication (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 332–337. IEEE Computer Society, 1989. `doi:10.1109/SFCS.1989.63499`.